

EXPRESS MAIL MAILING LABEL NUMBER EV318618446 US

PATENT
10830.0106.NPUS00

APPLICATION FOR UNITED STATES LETTERS PATENT

for

**METHOD AND APPARATUS FOR LOAD BALANCING OF
DISTRIBUTED PROCESING UNITS BASED ON PERFORMANCE METRICS**

by

Frank S. Caccavale

BACKGROUND OF THE INVENTION

1. Limited Copyright Waiver

[0001] A portion of the disclosure of this patent document contains computer code listings and command formats to which the claim of copyright protection is made. The copyright owner has no objection to the facsimile reproduction by any person of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office patent file or records, but reserves all other rights whatsoever.

2. Field of the Invention

[0002] The present invention relates generally to data processing networks, and more particularly to the load balancing of distributed processing units based on performance metrics.

3. Description of Related Art

[0003] It is often advantageous to distribute data processing tasks among a multiplicity of data processing units in a data processing network. The availability of more than one data processing unit provides redundancy for protection against processor failure. Data processing units can be added if and when needed to accommodate future demand. Individual data processing units can be replaced or upgraded with minimal disruption to ongoing data processing operations.

[0004] In a network providing distributed data processing, it is desirable to monitor distributed system performance. The performance of particular data processing units can be taken into consideration when configuring the network. Distributed performance can also be monitored to detect failures and system overload conditions. It

1 is desired to reduce system overload conditions without the cost of additional data
2 processing units.

3 SUMMARY OF THE INVENTION

4 [0005] In accordance with a first aspect, the invention provides a method of
5 operation in a data processing network including distributed processing units. The
6 method includes obtaining a respective utilization value of each distributed processing
7 unit, applying a mapping function to the respective utilization value of each distributed
8 processing unit to obtain a respective weight for each distributed processing unit, and
9 using the respective weights for the distributed processing units for distributing work
10 requests to the distributed processing units so that the respective weight for each
11 distributed processing unit specifies a respective frequency at which the work requests
12 are distributed to the distributed processing unit.

13 In accordance with another aspect, the invention provides a method of operation
14 in a data processing network including distributed processing units. The method includes
15 obtaining a respective utilization value of each distributed processing unit, applying a
16 mapping function to the respective utilization value of each distributed processing unit to
17 obtain a respective weight for each distributed processing unit, using the respective
18 weights for the distributed processing units for producing a distribution list for
19 distributing work requests to the distributed processing units for load balancing of the
20 work requests upon the processing units, and repetitively randomizing the distribution list
21 during the distribution of the work requests to the distributed processing units.

22 [0006] In accordance with yet another aspect, the invention provides a method
23 of operation in a data processing network including a network file server and a plurality

1 of virus checking servers. The method includes the network file server obtaining a
2 respective utilization value of each virus checking server, the respective utilization value
3 indicating a percentage of saturation of each virus checking server. The method further
4 includes the network file server applying a mapping function to the respective utilization
5 value to obtain a respective weight for each virus checking server, and the network file
6 server using the respective weights for the virus checking servers for weighted round-
7 robin load balancing of virus checking requests from the network file server to the virus
8 checking servers.

9 [0007] In accordance with still another aspect, the invention provides a data
10 processing system including distributed processing units and a processor coupled to the
11 distributed processing units for distributing work requests to the distributed processing
12 units. The processor is programmed for obtaining a respective utilization value of each
13 distributed processing unit, applying a mapping function to the respective utilization
14 value of each distributed processing unit to obtain a respective weight for each distributed
15 processing unit, and using the respective weights for the distributed processing units for
16 distributing work requests to the distributed processing units so that the respective weight
17 for each distributed processing unit specifies a respective frequency at which the work
18 requests are distributed to the distributed processing unit.

19 [0008] In accordance with yet another aspect, the invention provides a data
20 processing system including distributed processing units and a processor coupled to the
21 distributed processing units for distributing work requests to the distributed processing
22 units. The processor is programmed for obtaining a respective utilization value of each
23 distributed processing unit, applying a mapping function to the respective utilization

1 value of each distributed processing unit to obtain a respective weight for each distributed
2 processing unit, using the respective weights for the distributed processing units for
3 producing a distribution list for distributing work requests to the distributed processing
4 units for load balancing of the work requests upon the processing units, and repetitively
5 randomizing the distribution list during the distribution of the work requests to the
6 distributed processing units.

7 [0009] In accordance with a final aspect, the invention provides a data
8 processing system including virus checking servers and a network file server coupled to
9 the virus checking servers for distributing virus checking requests to the virus checking
10 servers. The network file server is programmed for obtaining a respective utilization
11 value of each virus checking server, the respective utilization value indicating a
12 percentage of saturation of each virus checking server, applying a mapping function to
13 the respective utilization value to obtain a respective weight for each virus checking
14 server, and using the respective weights for the virus checking servers for weighted
15 round-robin load balancing of virus checking requests from the network file server to the
16 virus checking servers.

18 BRIEF DESCRIPTION OF THE DRAWINGS

19 [00010] Other objects and advantages of the invention will become apparent
20 upon reading the detailed description with reference to the drawings, in which:

21 [00011] FIG. 1 is a block diagram of a data processing system incorporating
22 the present invention for load balancing of virus checkers;

1 **[00012]** FIG. 2 is a block diagram of an Internet site incorporating the present
2 invention for load balancing of Internet servers;

3 **[00013]** FIG. 3 is a flowchart of a method of using a virus checker in the
4 system of FIG. 1;

5 **[00014]** FIG. 4 is a block diagram showing details of a virus checker and an
6 event monitor in a server in the system of FIG. 1;

7 **[00015]** FIG. 5 is a flowchart of the operation of the event monitor

8 **[00016]** FIG. 6 shows a format that the event monitor could user for recording
9 statistics in a database in a file in the server of FIG. 4;

10 **[00017]** FIG. 7 is a graph of auto-correlation of server response time;

11 **[00018]** FIG. 8 shows a phase space of server response time;

12 **[00019]** FIG. 9 is a flowchart of a procedure for determining metric entropy of
13 virus checker performance;

14 **[00020]** FIGS. 10 to 12 comprise a flowchart of an analysis engine task for
15 evaluating virus checker performance statistics;

16 **[00021]** FIG. 13 shows a graph of response time as a function of server
17 workload;

18 **[00022]** FIG. 14 is a block diagram showing load balancing components in the
19 data processing system of FIG. 1;

20 **[00023]** FIG. 15 is a flowchart of a basic procedure for load balancing upon the
21 virus checkers based on the performance statistics;

1 **[00024]** FIGS. 16 and 17 comprise a flowchart of a first implementation of
2 basic procedure in FIG. 14, in which a distribution list for weighted round-robin load
3 balancing is obtained by a randomization method; and

4 **[00025]** FIG. 18 is a flowchart of a second implementation of the basic
5 procedure in FIG. 14, in which weighted round-robin load balancing is performed by a
6 deterministic method.

7 **[00026]** While the invention is susceptible to various modifications and
8 alternative forms, specific embodiments thereof have been shown by way of example in
9 the drawings and will be described in detail. It should be understood, however, that it is
10 not intended to limit the form of the invention to the particular forms shown, but on the
11 contrary, the intention is to cover all modifications, equivalents, and alternatives falling
12 within the scope of the invention as defined by the appended claims.

14 **DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS**

15 Collection of Performance Parameters from Distributed Processing Units

16 **[00027]** With reference to FIG. 1, there is shown a distributed data processing
17 system incorporating the present invention for load balancing of distributed processing
18 units. The data processing system includes a data network 21 interconnecting a number
19 of clients and servers. The data network 21 may include any one or more of network
20 connection technologies, such as Ethernet or Fibre Channel, and communication
21 protocols, such as TCP/IP or UDP. The clients include work stations 22 and 23. The
22 work stations, for example, are personal computers. The servers include conventional
23 Windows NT/2000 file servers 24, 25, 26, and a very large capacity network file server
24 27. The network file server 27 functions as a primary server storing files in nonvolatile

1 memory. The NT file servers 24, 25, 26 serve as secondary servers performing virus
2 checking upon file data obtained from the network file server 27. The network file server
3 27 is further described in Vahalia et al., U.S. Patent 5,893,140 issued April 6, 1999,
4 incorporated herein by reference. Such a very large capacity network file server 27 is
5 manufactured and sold by EMC Corporation, 176 South Street, Hopkinton, Mass. 01748.

6 [00028] The network file server 27 includes a cached disk array 28 and a
7 number of data movers 29, 30 and 31. The network file server 27 is managed as a
8 dedicated network appliance, integrated with popular network operating systems in a
9 way, which, other than its superior performance, is transparent to the end user. The
10 clustering of the data movers 29, 30, 31 as a front end to the cached disk array 28
11 provides parallelism and scalability. Each of the data movers 29, 30, 31 is a high-end
12 commodity computer, providing the highest performance appropriate for a data mover at
13 the lowest cost.

14 [00029] Each of the NT file servers 24, 25, 26 is programmed with a respective
15 conventional virus checker 32, 33, 34. The virus checkers are enterprise class anti-virus
16 engines, such as the NAI/McAfee's NetShield 4.5 for NT Server, Symantec Norton
17 AntiVirus 7.5 Corporate Edition for Windows NT, Trend Micro's ServerProtect 5.5 for
18 Windows NT Server. In each of the NT file servers 24, 25, 26, the virus checker 32, 33,
19 34 is invoked to scan a file in the file server in response to certain file access operations.
20 For example, when the file is opened for a user, the file is scanned prior to user access,
21 and when the file is closed, the file is scanned before permitting any other user to access
22 the file.

1 **[00030]** The network file server 27, however, is not programmed with a
2 conventional virus checker, because a conventional virus checker needs to run in the
3 environment of a conventional operating system. Network administrators, who are the
4 purchasers of the file servers, would like the network file server 27 to have a virus
5 checking capability similar to the virus checking provided in the conventional NT file
6 servers 24, 25, 26. Although a conventional virus checker could be modified to run in the
7 environment of the data mover operating system, or the data mover operating system
8 could be modified to support a conventional virus checker, it is advantageous for the
9 network file server 27 to use the virus checkers 27, 28, 29 in the NT file servers to check
10 files in the network file server 27 in response to user access of the files in the network file
11 server. This avoids the difficulties of porting a conventional virus checker to the network
12 file server, and maintaining a conventional virus checker in the data mover environment
13 of the network file server. Moreover, in many cases, the high-capacity network file
14 server 27 is added to an existing data processing system that already includes one or more
15 NT file servers including conventional virus checkers. In such a system, all of the files in
16 the NT file servers 24, 25, 26 can be migrated to the high-capacity network file server 27
17 in order to facilitate storage management. The NT file servers 24, 25, 26 in effect
18 become obsolete for data storage, yet they can still serve a useful function by providing
19 virus checking services to the network file server 27.

20 **[00031]** In general, when a client 22, 23 stores or modifies a file in the network
21 file server 27, the network file server determines when the file needs to be scanned.
22 When anti-virus scanning of a file has begun, other clients are blocked on any access to
23 that file, until the scan completes on the file. The network file server 27 selects a

1 particular one of the NT file servers 24, 25, 26 to perform the scan, in order to balance
2 loading upon the NT file servers for anti-virus scanning processes. The virus checker in
3 the selected NT file server performs a read-only access of the file to transfer file data
4 from the network file server to random access memory in the selected NT file server in
5 order to perform the anti-virus scan in the NT file server. Further details regarding the
6 construction and operation of the virus checkers 32, 33, 34 and the interface between the
7 virus checkers and the network file server 27 are found in Caccavale United States Patent
8 Application Publication No. US 2002/0129277 A1 published Sep. 12, 2002, incorporated
9 herein by reference.

10 [00032] In the system of FIG. 1, the NT file servers function as distributed
11 processing units for processing of anti-virus scans. It is desirable to determine a measure
12 of system performance, and trigger an alarm when the measure of system performance
13 indicates a presence of system degradation. For this purpose, the system includes a
14 service processor 36 programmed with an analysis engine application for collecting
15 performance parameters from the NT file servers, and for performing an analysis of these
16 performance parameters. The service processor 36 could be a processor in any one of the
17 client terminals 22, 23 or the file servers 24, 25, 26, and 27 in the system of FIG. 1. For
18 example, the service processor could be the processor of the client terminal of a system
19 administrator for the system of FIG. 1. It is also desirable to perform load balancing
20 upon the NT file servers based on the performance parameters from the NT file servers,
21 as will be further described below with reference to FIGS. 14 to 18.

22 [00033] With reference to FIG. 2, there is shown another example of a data
23 processing system in which the present invention can be used. FIG. 2 shows the Internet

1 40 connecting clients 41 and 42 to a gateway router 43 of an Internet site. The Internet
2 site includes Internet servers 45, 46, and 47 and a service processor 48 programmed with
3 an analysis engine application 49. In this example, the gateway router 43 receives client
4 requests for access to a “web page” at the Internet address of the gateway router. The
5 gateway router 43 performs load balancing by routing each client request to a selected
6 one of the Internet servers 45, 46, and 47. The Internet servers function as distributed
7 data processing units. The analysis engine application 49 collects performance
8 parameters from the Internet servers 45, 46, and 47 in order to determine a measure of
9 system performance, and to trigger an alarm when the measure of system performance
10 indicates a presence of system degradation. The analysis engine application 49 in the
11 system of FIG. 2 operates in a fashion similar to the analysis engine application 35 and
12 FIG. 1. The performance parameters can also be used to perform load balancing upon the
13 Internet servers 45, 46, and 47, in a fashion similar to the load balancing described below
14 with reference to FIGS. 14 to 18.

15 [00034] With reference to FIG. 3, there is shown a flowchart of a method of
16 using a virus checker in the system of FIG. 1. In a first step 50 of FIG. 3, a client (22, 23
17 in FIG. 1) sends new data to the primary network file server (27 in FIG. 1). Next, in step
18 51, the primary network file server receives the new data, and selects one of the virus
19 checkers for load balancing. For example, a virus checker is selected using a “round
20 robin” method that places substantially the same workload upon each of the virus
21 checkers. In step 52, the primary network file server sends an anti-virus scan request to
22 the NT file server (24, 25, or 26 in FIG. 1) having the selected virus checker (32, 33, or
23 34). The scan request identifies the new data to be scanned. In step 53, the selected virus

1 checker receives the scan request, and accesses the new data in the primary network file
2 server. In step 54, the selected virus checker determines if there is a risk of a virus being
3 present in the new data, and recommends an action if there is a risk of a virus being
4 present.

5 **[00035]** FIG. 4 shows details of a virus checker 32 and an event monitor 85 in
6 an NT server 24 in the system of FIG. 1. A scan request filter 81 receives a scan request
7 from the primary network file server (27 in FIG. 1). The scan request filter 81 determines
8 if the virus checker will accept the request. Once a request is accepted, the scan request
9 filter 81 passes the request to an event driver 82. The event driver 82 sends a “begin”
10 event signal to an event monitor and statistics generator 85, which is an application
11 program in the NT file server 24. The event monitor and statistics generator 85 responds
12 to the “begin” event signal by recording the time of acceptance of the scan request. The
13 event driver 82 passes the scan request to a virus scanner 83.

14 **[00036]** The virus scanner 83 obtains the new data from the primary network
15 file server (27 in FIG. 1) and scans that data for potential virus contamination. Upon
16 completion of the scan of the data, the results are passed to an event driver 84. The event
17 driver 84 sends an “end” event signal to the event monitor and statistics generator 85.
18 The event driver 82 and the event driver 84 may use a common interface routine in the
19 virus checker 32, in order to interface with the event monitor and statistics generator 85.
20 After the event driver 84 sends the “end” event signal to the event monitor and statistics
21 generator 85, the virus checker 32 returns an acknowledgement to the primary network
22 file server indicating the result of completion of the anti-virus scan.

1 **[00037]** The event monitor and statistics generator 85 responds to the “end”
2 event signal by obtaining the time of the “end” event and computing the duration of time
3 between the corresponding “begin” event and the “end” event. Moreover, during each
4 test interval (denoted as T_i), all response times are stored and an average is taken. The
5 average response time for the test interval, and the total number of scan requests
6 processed by the virus scanner 83 during the test interval, are stored in the “Windows
7 Management Instrumentation” (WMI) data base 86 maintained by the Microsoft
8 Corporation WINDOWS operating system 87 of the NT file server 24. After storage of
9 the average response time and the total number of scan requests, a new test interval is
10 started and new response times are stored for use in the next average response time
11 generation. For example, the default setting for the test interval is 10 seconds, and the
12 number of consecutive test interval results stored in the WMI database is 30 or greater.

13 **[00038]** The use of the event monitor 85 in the NT file server 24 to compute
14 and store averages of the response time over the test intervals reduces the total data set
15 that need be analyzed. Therefore, the storage of the data in the WMI 86 is more compact,
16 network resources are conserved when the analysis engine accesses the WMI, and
17 processing requirements of the analysis engine are reduced. The use of the WMI as an
18 interface between the event monitor and the analysis engine ensures that the event
19 monitor 85 need not know anything about the protocol used by the analysis engine to
20 access the WMI. The WMI provides a standard data storage object and internal and
21 external access protocols that are available whenever the Windows operating system 87 is
22 up and running.

1 **[00039]** FIG. 5 is a flowchart of the operation of the event monitor and
2 statistics generator (85 in FIG. 4). In a first step 91, in response to a “begin” event,
3 execution branches to step 92. In step 92, the event monitor records the time of the
4 “begin” event for the scan, and processing for the “begin” event is finished. If the event
5 monitor is responding to something other than a “begin” event, execution continues from
6 step 91 to step 93.

7 **[00040]** In step 93, in response to an “end” event, execution branches from step
8 93 to step 94. In step 94, the event monitor computes the response time for the scan as
9 the difference between the time of the end event and the time of the begin event. Then in
10 step 95, the event monitor records the response time for the scan, and processing for the
11 “end” event is finished. If the event monitor is responding to something other than a
12 “begin” event or an “end” event, execution continues from step 93 to step 96.

13 **[00041]** In step 96, in response to the end of a test interval, execution branches
14 from step 96 to step 97. In step 97, the event monitor computes the number of requests
15 processed during this test interval, and the average response time. The average response
16 time is computed as the sum of the response times recorded in step 95 during this test
17 interval, divided by the number of requests processed during this test interval. Then in
18 step 98, the event monitor records the number of requests processed during this test
19 interval, and the average response time. After step 98, processing is finished for the end
20 of the test interval.

21 **[00042]** In the procedure of FIG. 5, the processing of a request may begin in
22 one test interval and be completed in a following test interval. In this situation, the
23 number of requests processed (NR) for a particular test interval indicates the number of

1 requests completed in that test interval. Also, it is possible for a “running sum” of the
2 response times and a “running sum” of the number of requests processed to be
3 accumulated and recorded in step 95, instead of simply recording the response time in
4 step 95. In this case, the running sum of the number of requests processed will be the
5 total number of requests completed over the ending test interval when step 97 is reached,
6 and the average response time for the ending test interval can be computed in step 97 by
7 dividing the running sum of the response times by this total number of requests
8 completed over the ending test interval. Then in step 98, after recording the number of
9 requests processed and the average response time, the running sum of the response times
10 and the running sum of the number of requests processed can be cleared for accumulation
11 of the response times and the number of requests processed during the next test interval.

12 **[00043]** FIG. 6 shows a format that the event monitor could use for recording
13 statistics in a database stored in a file in the server of FIG. 4. The database is in the form
14 of an array or table 100. For example, the table 100 includes thirty-two rows. Included
15 in each row is an index, the response time (RT), and the number of requests processed
16 (NR) for the test interval indicated by the index. The index is incremented each time a
17 new row of data is written to the table 100. The row number of the table is specified by
18 the least significant five bits of the index. The last time a row of data was written to the
19 table can be determined by searching the table to find the row having the maximum value
20 for the index. This row will usually contain the row of data that was last written to the
21 table, unless the maximum value for the index is its maximum possible value and it is
22 followed by a row having an index of zero, indicating “roll-over” of the index has
23 occurred. If “roll-over” has occurred, then the last time a row of data was written to the

1 table occurred for the row having the largest index that is less than 32. By reading the
2 table 100 in a server, the analysis engine application in the service processor can
3 determine the index for the most recent test interval and copy the data from a certain
4 number (N) of the rows for the most recent test intervals into a local array in the service
5 processor.

6 **[00044]** In a preferred implementation, Microsoft WMI services are used to
7 define a data structure for the statistics in the WMI database (86 in FIG. 4), to put new
8 values for the statistics into the data structure, and to retrieve the new values of the
9 statistics from the data structure. In general, WMI is a Microsoft Corporation
10 implementation of WBEM. WBEM is an open initiative that specifies how components
11 can provide unified enterprise management. WBEM is a set of standards that use the
12 Common Information Model (CIM) for defining data, xmlCIM for encoding data, and
13 CIM over Hyper-Text Transmission Protocol (HTTP) for transporting data. An
14 application in a data processing unit uses a WMI driver to define a data structure in the
15 WMI database and to put data into that data structure in the WMI database. User-mode
16 WMI clients can access the data in the WMI database by using WMI Query language
17 (WQL). WQL is based on ANSI Standard Query Language (SQL).

18 **[00045]** In the preferred implementation, the data structure in the WMI
19 database stores the total files scanned (NR) by the virus checker, the average response
20 time (RT) per scan, the saturation level for the average response time per scan, state
21 information of the virus checker, and state information of the event monitor and statistics
22 generator. A WMI provider dll sets and gets data to and from the WMI database.

1 **[00046]** The following is an example of how the WMI provider dll is used to
2 put data into the WMI database:

```
3
4  STDMETHODCALLTYPE CCAVAProvider::PutProperty(    long lFlags,
5                                                    const BSTR Locale,
6                                                    const BSTR InstMapping,
7                                                    const BSTR PropMapping,
8                                                    Const VARIANT *pvValue)
9  {
10      if(!_wcsicmp(PropMapping, L"ScansPerSec"))
11      {
12          m_dScansPerSec = pvValue->dblVal;
13      }
14  }
```

15

16 **[00047]** The following is an example of how the WMI provider dll is used to
17 get data from the WMI database:

18

```
19  STDMETHODCALLTYPE CCAVAProvider::GetProperty( long lFlags,
20                                                    const BSTR Locale,
21                                                    const BSTR InstMapping,
22                                                    const BSTR PropMapping,
23                                                    VARIANT *pvValue)
24  {
25      if(!_wcsicmp(PropMapping, L"ScansPerSec"))
26      {
27          pvValue->vt = VT_R8;
28          pvValue->dblVal = m_dScansPerSec;
```

```

1          }
2
3          return sc;
4      }

```

6 **[00048]** The following is an example of how the event monitor sets its
 7 processed results in the WMI database via the provider dll for transmission to the analysis
 8 engine:

```

10          // this object will time the scan
11          CScanWatcher* pSW = new CScanWatcher;
12          // this is the scan of the file
13          VC_Status s = m_pVAgent->CheckFile(csFileName);
14
15          // the destructor of the object completes the calculations
16       and records the scan stats
17          delete pSW;

```

19 **[00049]** The following is an example of how the analysis engine (e.g., a Visual
 20 Basic GUI application) may use the WMI provider dll to retrieve the statistics from the
 21 WMI database and present the statistics to a user:

```

23          Dim CAVA As SWbemObject
24          Dim CAVASet As SWbemObjectSet
25          Dim CurrentCAVA As SWbemObject
26          Dim strServer As String
27          Dim strE

```

```

1
2         Open ".\cavamon.dat" For Input As #1      'open dat file
3
4         lstStatsOutput.Clear                      'clear output
5
6         Do While Not EOF(1) ' for each machine in cavamon.dat
7
8             Input #1, strServer
9             If strServer = "" Then
10                 GoTo NextLoop
11             Else
12                 On Error GoTo ErrorHandler
13             End If
14
15             'Debug.Print strServer
16             Set Namespace = GetObject("winmgmts://" & strServer &
17 "/root/emc")
18             'this will trap a junk server name
19             If Err.Number <> 0 Then GoTo NextLoop
20
21             Set CAVASet = Namespace.InstancesOf("CAVA")
22
23             For Each CAVA In CAVASet      ' for each cava in a given machine
24                 ' DISPLAY EACH CAVA'S WMI INFO
25                 'Set          CurrentCAVA          =
26 GetObject("winmgmts:" & CAVA.Path_.RelPath)
27
28                 lstStatsOutput.AddItem ("Server: \" & strServer & "\")
29

```

```

1         lstStatsOutput.AddItem ("---Cumulative Statistics---")
2
3         If Not IsNull(CAVA.AVEngineState) Then
4             lstStatsOutput.AddItem ("AV Engine State: " &
5 CAVA.AVEngineState)
6         End If
7
8         If Not IsNull(CAVA.AVEngineType) Then
9             lstStatsOutput.AddItem ("AV Engine Type: " &
10 CAVA.AVEngineType)
11         End If
12
13         If Not IsNull(CAVA.FilesScanned) Then
14             lstStatsOutput.AddItem ("Total Files Scanned: " &
15 CAVA.FilesScanned)
16         End If
17
18         lstStatsOutput.AddItem ("---Interval Statistics---")
19
20         If Not IsNull(CAVA.Health) Then
21             lstStatsOutput.AddItem ("          AV Health: " &
22 CAVA.Health)
23         End If
24
25         If Not IsNull(CAVA.MilliSecsPerScan) Then
26             lstStatsOutput.AddItem ("          Milliseconds per Scan: "
27 & FormatNumber(CAVA.MilliSecsPerScan, 2))
28         End If
29

```

```

1           If Not IsNull(CAVA.SaturationPercent) Then
2               If CAVA.SaturationPercent = 0 Then
3                   lstStatsOutput.AddItem ("      Saturation %: N/A")
4               Else
5                   lstStatsOutput.AddItem ("      Saturation %: " &
6 FormatNumber((CAVA.SaturationPercent * 100), 2))
7               End If
8           End If
9
10          If Not IsNull(CAVA.ScansPerSec) Then
11              lstStatsOutput.AddItem ("      Scans Per Second: " &
12 CAVA.ScansPerSec)
13          End If
14
15          If Not IsNull(CAVA.State) Then
16              lstStatsOutput.AddItem ("      CAVA State: " &
17 CAVA.State)
18          End If
19
20          If Not IsNull(CAVA.Version) Then
21              lstStatsOutput.AddItem ("      CAVA Version: " &
22 CAVA.Version)
23          End If
24
25          lstStatsOutput.AddItem ("")
26
27          Next 'for each cava in a given machine
28      NextLoop:
29          Loop ' for each machine in cavamon.dat

```

```

1         Close #1      'close opened file
2         GoTo SuccessHandler
3
4     ErrorHandler:
5
6         Close #1
7
8         tmrStats.Enabled = False      'disable the timer
9
10        cmdStats.Caption = "Get Stats" 'change button caption
11
12        MsgBox "An error has occurred: " & Err.Description
13
14
15    SuccessHandler:
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
268
```

$$\Delta RT_{ij} = RT_{ij} - RT_{(i-1)j}$$

If any virus checker exhibits exponential growth in the response time, as further described below with reference to FIG. 13, then an alarm is posted.

[00052] In order to reduce the overhead for computing, storing, and transporting the performance statistics over the network, it is desired for the test interval to include multiple scans, but the test interval should not have a duration that is so long that pertinent information would be lost from the statistics. For example, the computation of metric entropy, as described below, extracts information about a degree of correlation of the response times at adjacent test intervals. Degradation and disorder of the system is indicated when there is a decrease in this correlation. The duration of the test interval should not be so long that the response times at adjacent test intervals become substantially uncorrelated under normal conditions.

[00053] FIG. 7, for example, includes a graph of the auto-correlation coefficient (r) of the server response time RT . The auto-correlation coefficient is defined as:

$$r = \frac{\text{cov}(RT_t, RT_{t+\Delta t})}{\sigma_{RT}^2}$$

The value of the auto-correlation coefficient of the server response time RT ranges from 1 at $\Delta t=0$ to zero at $\Delta t=\infty$. Of particular interest is the value of time (T_{corr}) at which the auto-correlation coefficient has a value of one-half. System degradation and disorder in the server response time causes the graph of the auto-correlation coefficient to shift from the solid line position 101 to the dashed line position 102 in FIG. 7. This shift causes a most noticeable change in the value of auto-correlation coefficient for a Δt on the order of T_{corr} . Consequently, for extracting auto-correlation statistics or computing a metric

1 entropy by using a two-dimensional phase space, as further described below, the test
2 interval should be no greater than about T_{corr} .

3 **[00054]** The anti-virus scan tasks have the characteristic that each task requires
4 substantially the same amount of data to be scanned. If the scan tasks did not inherently
5 have this property, then each scan task could be broken down into sub-tasks each
6 requiring substantially the same processing time under normal conditions, in order to
7 apply the following analysis to the performance statistics of the sub-tasks. Alternatively,
8 the performance statistics for each task could be normalized in terms of the processing
9 time required for a certain number of server operations, such as scanning a megabyte of
10 data.

11 **[00055]** If the response times of the virus checkers vary randomly over the
12 possible ranges of response times, then the response time is becoming unpredictable and
13 there is a problem with the system. Similarly, if there is normally a substantial degree of
14 auto-correlation of the response time of each virus checker between adjacent test intervals
15 but there is a loss of this degree of auto-correlation, then the response time is becoming
16 unpredictable and there is a problem with the system. Assumptions about whether the
17 system is properly configured to handle peak loads are likely to become incorrect. Load
18 balancing methods may fail to respond to servers that experience a sudden loss in
19 performance. In any event, gains in performance in one part of the system may no longer
20 compensate for loss in performance in other parts of the system.

21 **[00056]** The unpredictability in the system can be expressed numerically in
22 terms of a metric entropy. The adjective “metric” denotes that the metric entropy has a

1 minimum value of zero for the case of zero disorder. For example, metric entropy for a
2 sequence of bits has been defined by the following equation:

$$H_{\mu} = -\frac{1}{L} \sum_{i=1}^n p_i \log_2 p_i$$

3
4
5 where L is word length in the sequence, and p is the probability of occurrence for the i-th
6 L-word in the sequence. This metric entropy is zero for constant sequences, increases
7 monotonically when the disorder of the sequence increases, and reaches a maximum of 1
8 for equally distributed random sequences.

9
10 **[00057]** To compute a metric entropy for the entire system of virus checkers,
11 the analysis engine retrieves the response time arrays from the WMI databases of the NT
12 servers containing the virus checkers. These responses are averaged on a per interval
13 basis. Calling the response time from anti-virus engine (j) in test interval (i) $R_{t_{ij}}$, the
14 average is taken for across all of N anti-virus engines as:

$$RT_{avg(i)} = (\sum_{j=1}^N RT_{ij}) / N$$

15
16
17 Thus, the symbol $RT_{avg(i)}$ indicates the average response time across the N anti-virus
18 engines during the test interval (i).

19
20 **[00058]** Next a two-dimensional phase space is generated and cells in the phase
21 space are populated based upon the adjacent pairs of values in the $RT_{avg(i)}$ table. FIG. 8
22 shows an example of such a phase space 103. The worst case response time (the
23 saturation response time for the system of anti-virus engines) is divided by the resolution
24 desired per axis in order to determine the number of cells per axis. An example would be

1 if the saturation response were 800 ms (milliseconds) and the desired resolution per axis
2 were 10 ms then each axis of the phase space would consist of 10 ms intervals for a
3 duration of 80 intervals. This would give an 80 by 80 grid consisting of 6400 cells. The
4 desired resolution per axis, for example, is selected to be no greater than the standard
5 deviation of the average response time over the virus checkers for each test interval
6 during normal operation of the system.

7 **[00059]** Each pair of adjacent values of $RT_{avg(i)}$ is analyzed to determine the
8 location in the phase space that this pair would occupy. An example would be if two
9 adjacent values were 47.3 ms and 98 ms. This would mean that on the first axis of the
10 phase space the interval is the 5th interval (i.e. from 40 ms to 50 ms) and the second axis
11 is the 10th interval (i.e. from 90 ms to 100 ms). This pair of values would represent an
12 entry to the (5,10) cell location in the phase space. As pairs of values are analyzed the
13 number of entries in each phase space cell location is incremented as entries are made.

14 **[00060]** The worst case metric entropy would occur if every cell location were
15 entered with equal probability. The value of this worst case metric entropy is given by
16 the formula:

$$17 \quad \quad \quad -1 \times (\log_{10}(\text{worst case probability of random entry}))$$

19

20 **[00061]** In the example of an 80 by 80 interval phase space the worst case
21 probability would be (1) out of (6400) so the value of the worst case metric entropy
22 would be approximately 3.81. The best case metric entropy should be zero. This would
23 indicate that all entries always have the same response time.

1 **[00062]** To approximate the metric entropy function the probabilities of the
2 entries are put into the formula:

$$3 \quad - \sum_{i=1}^{80} \sum_{j=1}^{80} \text{Pr}_{ij} (\log_{10} \text{Pr}_{ij})$$

4
5 In this formula Pr_{ij} is the probability of the $(ij)^{\text{th}}$ phase space cell location being hit by an
6 entry based on the entries accumulated. Should all entries accumulate in a single phase
7 space cell location then the probability of that cell location being hit is (1) and then $\log(1)$
8 is zero hence the approximated metric entropy is zero, the best case metric entropy.
9 Should all entries be evenly dispersed across all possible phase space cell locations then
10 the summation returns the probability of each phase space location as $1/(\text{total number of}$
11 $\text{phase space locations})$. Since there are the as many terms in the sum as there are phase
12 space cell locations then sum becomes:

$$13 \quad -1 \times (\text{phase space area}) \times (1/(\text{phase space area})) \times \log_{10} \text{Pr}_{ij}$$

14
15
16 where the phase space area is the total number of cell locations in the phase space (6400
17 in the example given above). Since the PR_{ij} is the worst case probability this becomes
18 the same value as the worst case metric entropy. Therefore the metric entropy from this
19 computation ranges from 0 to about 3.81. This matches, to a proportionality constant, the
20 metric entropy as defined in the literature, which ranges from 0 to 1.

21 **[00063]** Although the metric entropy from this computation could be
22 normalized to a maximum value of 1, there is no need for such a normalization, because
23 the metric entropy from this computation is compared to a specified maximum limit to

1 trigger an alarm signaling system degradation. Therefore, there is a reduction in the
2 computational requirements compared to the computation of true metric entropy as
3 defined in the literature. Computations are saved in the analysis engine so that the
4 analysis engine can operate along with other applications without degrading performance.

5 **[00064]** FIG. 9 shows the steps introduced above for computing a metric
6 entropy for the virus checker system of FIG. 1. In a first step 111, an average response
7 time of each virus checker is computed over each test interval in a sequence of test
8 intervals. For example, an event monitor in a server for each virus checker computes the
9 average response time of each virus checker for each test interval. In step 112, the
10 average response time over all of the virus checkers is computed for each test interval.
11 For example, the analysis engine application computes the average response time over all
12 of the virus checkers for each test interval.

13 **[00065]** In step 113, a two-dimensional array of occurrence accumulators is
14 cleared. Each accumulator corresponds to a respective cell in the two-dimensional phase
15 space. In step 114, for each pair of adjacent average response times over all of the virus
16 checkers, the two response times are quantized to obtain a pair of indices indexing one of
17 the accumulators, and the indexed accumulator is incremented, thereby producing a
18 histogram over the two-dimensional phase space. In step 115, the metric entropy for the
19 system is computed from this histogram. For example, the analysis engine application
20 performs steps 113, 114, and 115.

21 **[00066]** The value computed for the metric entropy is compared to a specified
22 maximum limit. When the specified maximum limit is exceeded, an alarm is posted
23 notifying that the behavior of the system is becoming erratic. The system can be

1 investigated to determine if the load is being improperly distributed (possibly due to a
2 malfunctioning virus checker or an improper configuration of the network).

3 **[00067]** As new values of metric entropy are accumulated a rate of change is
4 calculated between adjacent time intervals according to:

$$\Delta H_i = H_{(i)} - H_{(i-1)}$$

6 The rate of change is checked for an exponential rate of increase. This rate of change can
7 signal that there are changes occurring in the system that will lead to a problem. This
8 measurement is a form of predictive analysis that can notify system users that a problem
9 can be averted if action is taken.

10 **[00068]** The utilization of individual virus checkers is computed based on the
11 data retrieved from the WMI database in each of the NT file servers. The data include
12 the response time values (RT_{ij}) and the number of requests for scans (NR_{ij}) during the
13 (i_{th}) test interval for the (j_{th}) virus checker. The interval duration (τ) divided by the
14 number of requests (NR_{ij}) yields the average time between requests. The reciprocal of
15 the average time between requests is the request rate. The response time (i.e. RT_{ij})
16 divided by the average time between requests gives the utilization of that virus checker
17 during that interval. Therefore, the utilization (α_{ij}) of the (j_{th}) virus checker over the (i_{th})
18 test interval is computed as:

$$\alpha_{ij} = (RT_{ij}) (NR_{ij}) / (\tau)$$

22 **[00069]** A maximum limit (for example 60%) is set for virus checker
23 utilization. If this maximum limit is exceeded for a virus checker then that virus checker

1 is over utilized. A recommendation is made based on the virus checker utilization for
2 corrective action. Should a single virus checker be over utilized then there is an
3 imbalance in the system and the configuration should be corrected. Should all utilization
4 values be high a recommendation is made on the number of virus checkers that should be
5 added to the system. An additional NT file server is added with each additional virus
6 checker.

7 [00070] The utilization of the entire set of virus checkers can be approximated
8 by computing an average number of requests across the virus checkers according to:

9

$$10 \quad NR_{avg(i)} = \left(\sum_{j=1}^N NR_{ij} \right) / N$$

11 and then using the average response time across the virus checkers ($RT_{avg(i)}$) and the
12 average number of requests across the virus checkers ($NR_{avg(i)}$) in the formula for
13 utilization, according to:

14

$$15 \quad \alpha_{avg(i)} = (RT_{avg(i)})(NR_{avg(i)}) / (\tau)$$

16

17 The values of $\alpha_{avg(i)}$ for several test intervals are accumulated, and averaged across a
18 series of adjacent test intervals, to remove irregularities. As values of the utilization are
19 accumulated, a rate of change of the utilization between adjacent test intervals is
20 computed, accumulated, and continually analyzed. Should the rate become exponentially
21 increasing then an alarm is given indicating a possible problem. This is a predictive
22 analysis function just as was done for metric entropy.

1 **[00071]** Dividing the actual utilization by the desired utilization (for example
2 60%) yields the factor that the number of virus checkers must be multiplied by to reach a
3 utilization that matches the desired utilization number. A rounding function is done on
4 the product of the utilization factor and the number of virus checkers to produce the
5 recommended number of virus checkers to achieve a desired utilization.

6 **[00072]** The estimation technique used to determine the recommended number
7 of virus checkers is as follows:

8 **[00073]** Call the average response rate μ , the average workload λ , the desired
9 utilization ρ and the number of virus checkers servicing the load M , then the formula

$$N = M(\lambda/\mu\rho)$$

13 gives the approximation used for the recommended number of virus checkers. In this
14 case the average workload is in terms of a number of scans per test interval, and the
15 average response rate, in responses per second, is the reciprocal of the computed $RT_{avg(i)}$.
16 An example would be:

17 **[00074]** Given one virus checker being analyzed with an average workload of
18 10 scans per second and an average response time of 0.2 seconds per request and the
19 desired utilization being 60% then the formula results in the desired number of virus
20 checkers (N) being 3 1/3 virus checkers. This is rounded up to 4 as the recommended
21 number of virus checkers. If the analysis had been done on a group of 3 virus checkers,
22 with the numbers given above, then the recommended number of virus checkers would
23 have become 3(3 1/3) or 10 virus checkers total.

1 **[00075]** The desired value of utilization is a programmable number and the
2 value of 60% has been assumed for the virus checker examples above.

3 **[00076]** FIGS. 10 to 12 show a flowchart of an analysis engine task for
4 performing the analysis described above. This task is performed once for each test
5 interval. In the first step 121, the analysis engine gets the new values of the average
6 response time (RT) and number of requests (NR) for each virus checker from the
7 Windows Management Instrumentation (WMI) database of each virus checker server.
8 Next, in step 122, the analysis engine compares the average response time of each virus
9 checker to a predetermined limit (LIMIT1). If the limit is exceeded, then execution
10 branches from step 123 to step 124 to report the slow virus checker to the system
11 administrator to correct the virus checker or reconfigure the network. Execution
12 continues from step 124 to step 125. If the limit is not exceeded, execution continues
13 from step 123 to step 125.

14 **[00077]** In step 125, the analysis engine computes the rate of change in the
15 average response time of each virus checker. In step 126, if there is an exponential
16 increase in the average response time, then execution branches from step 126 to step 127
17 to report impending system degradation to the system administrator. (The detection of an
18 exponential increase will be further described below with reference to FIG. 13.)
19 Execution continues from step 127 to step 131 in FIG. 11. If there is not an exponential
20 increase, execution continues from step 126 to step 131 of FIG. 11.

21 **[00078]** In step 131 of FIG. 11, the analysis engine task computes metric
22 entropy for the system. In step 132, if the metric entropy exceeds a specified limit
23 (LIMIT2), then execution branches to step 133 to report instability of the system to the

1 system administrator to correct the virus checkers or reconfigure the network. Execution
2 continues from step 133 to step 134. If the limit is not exceeded, then execution
3 continues from step 132 to step 134.

4 **[00079]** In step 134, the analysis engine task computes the rate of change in the
5 metric entropy for the system. In step 135, if there is an exponential increase in the
6 metric entropy for the system, then execution branches to step 136 to report impending
7 system degradation to the system administrator and users. After step 136, execution
8 continues to step 141 of FIG. 12. If there is not an exponential increase, execution
9 continues from step 135 to step 141 of FIG. 12.

10 **[00080]** In step 141, the analysis engine computes the system utilization. In
11 step 142, if the system utilization exceeds a specified limit (LIMIT3), then execution
12 branches to step 143 to report the excessive system utilization to the system administrator
13 to reconfigure the system or to add additional servers. Execution continues from step 143
14 to step 144. In step 144, the analysis engine task computes and reports to the system
15 administrator the number of additional servers needed to achieve the desired utilization.
16 After step 144, the analysis engine task is finished for the current test interval. The
17 analysis engine task is also finished if the system utilization does not exceed the specified
18 limit in step 142.

19 **[00081]** FIG. 13 shows a graph 150 of response time (RT) as a function of
20 server workload (W). This graph exhibits a characteristic exponential increase in
21 response time once the response time reaches a threshold (TH) at the so-called “knee” or
22 saturation point 151 of the curve. One way of detecting the region of exponential
23 increase is to temporarily overload the server into the exponential region in order to

1 empirically measure the response time as a function of the workload. Once the response
2 time as a function of workload is plotted, the knee of the curve and the threshold (TH)
3 can be identified visually.

4 **[00082]** The knee 151 of the curve in FIG. 13 can also be located by the
5 following computational procedure, given that the curve is defined by N
6 workload/response time pairs (W_i, RT_i) for $i = 1$ to N. Calculating an average slope:

7
$$m_{avg} = (W_n - W_1) / (RT_n - RT_1);$$

8 and then calculate n-2 local slopes, $m_2 - m_{n-1}$, where

9
$$m_2 = (W_3 - W_1) / (RT_3 - RT_1)$$

10 and

11
$$m_{n-1} = (W_n - W_{n-2}) / (RT_n - RT_{n-2})$$

12 The knee of the curve is the one of the n points, x, which satisfies each of the following
13 conditions $m_x = m_{avg} \pm .5\%$; $m_{x-1} \leq m_{avg}$; and $m_{x+1} > m_{avg}$.

14 **[00083]** Once the threshold is identified, operation in the exponential region
15 can be detected by comparing the response time to the threshold (TH). In a similar
16 fashion, it is possible to detect an exponential increase in the rate of change of the
17 average response time or metric entropy by comparing the rate of change to a threshold
18 indicative of entry into an exponential region. In general, the alarm limits for the
19 measurements and performance statistics are programmable so that they can be tuned to
20 the type of server carrying the workload.

21

Load Balancing of Distributed Processing Units Based on Performance Metrics

[00084] It is desired to reduce the occurrence of overload of the virus checkers in the system of FIG. 1 without the cost of additional virus checkers. Each data mover can do this by performing the load balancing of step 51 in FIG. 3 based on the collected performance statistics. For example, as shown in FIG. 14, each NT file server 24, 25, 26 executes a respective program 155, 156, 157 for sensing the percentage utilization $\alpha[i]$ of the NT file server. This can be done as described above with reference to the event monitor and statistics generator 84 of FIG. 4. A weighted round-robin load balancing program 158 in the data mover 30 periodically collects the utilizations $\alpha[i]$ from the NT file servers, and accesses a mapping table 159 for converting the utilizations $\alpha[i]$ to a respective set of weights $W[i]$. The weighted round-robin load balancing program 158 uses the weights $W[i]$ for distributing the virus scan requests to the virus checkers 32, 33, 34 in the NT file servers 24, 25, and 26.

[00085] FIG. 15 shows the load balancing procedure used in the system of FIG. 14. In a first step 161 of FIG. 15, each virus checker (i) computes a utilization $\alpha[i]$ indicating the loading on the virus checker (i) as a percentage of the saturation level of the virus checker. Next, in step 162, the data mover collects the utilization $\alpha[i]$ from each of the virus checkers. For example, the data mover periodically sends a heartbeat signal to the virus checkers to check the health of the virus checkers. A system administrator can program the frequency of the heartbeat signal. In response to the heartbeat signal, each virus checker calculates its scanning utilization $\alpha[i]$, and each virus checker returns its scanning utilization in its heartbeat response.

1 **[00086]** In step 163, the data mover applies a mapping function to convert each
2 utilization $\alpha[i]$ to a respective weight $W[i]$ estimated to cause a balancing of the loading
3 upon the virus checkers. In other words, the weights are estimated so that if they are
4 representative of the respective workloads placed on the virus checkers, then the
5 utilization $\alpha[i]$ is expected to become substantially uniform across the virus checkers. In
6 step 164, the data mover uses the weights $W[i]$ for weighted round-robin load balancing
7 of the anti-virus scan requests from the data mover to the virus checkers. In other words,
8 the anti-virus scan requests are assigned to the virus checkers in such a way that the
9 weights $W[i]$ specify the respective workloads placed on the virus checkers. The weights
10 $W[i]$ are used for load balancing in step 164 until a next heartbeat interval occurs. In step
11 165, when the next heartbeat interval occurs, execution loops back to step 161.

12 **[00087]** The load balancing method of FIG. 15 can be heuristic in the sense
13 that the estimation of the respective weights $W[i]$ from the utilizations $\alpha[i]$ may take into
14 account a number of factors that affect the performance of the virus checkers even though
15 the effect of each factor may not be known precisely. In general, the respective weight
16 $W[i]$ for the virus checker (i) should vary inversely with respect to the utilization $\alpha[i]$ of
17 the virus checker (i). If the virus checkers are not identical, then the respective weight
18 $W[i]$ for the virus checker (i) should be proportional to the processing capability of the
19 virus checker (i) relative to the other virus checkers.

20 **[00088]** The estimation of the respective weights $W[i]$ should also take into
21 consideration the fact that load balancing based on the utilization $\alpha[i]$ creates a feedback
22 loop in which the open-loop gain is proportional to the incremental change in the
23 respective weight $W[i]$ for an incremental change in the utilization $\alpha[i]$ for the virus

1 checker (i). It is possible that a relatively high open loop gain may lead to an undesired
2 overshoot in the correction of an unbalanced condition, and at worst instability in the
3 form of ringing or oscillation in the utilization. One would like a nearly critically
4 damped condition in which an imbalance in the utilization over the virus checkers is
5 corrected as much as possible in the next heartbeat interval without ringing in subsequent
6 heartbeat intervals. Therefore, in practice, one may start with a mapping function based
7 on known or estimated performance capabilities of the virus checkers, and then modify
8 that mapping function based on observing how the utilization changes over a series of the
9 heartbeat intervals in response to an imbalance in utilization.

10 [00089] For the case of identical virus checkers, the following mapping
11 function has been used:

12

13	<u>Utilization (α)</u>	<u>Weight (W)</u>
14	0-40%	4
15	41-60%	3
16	61-90%	2
17	91-100%	1

18

19 [00090] The above mapping function, for example, is programmed into a table
20 (159 in FIG. 14), so that the utilization ranges and the weights are configurable. If a
21 system would have different types of virus checkers, for example having different
22 processing capabilities or operating environments, a respective mapping table could be
23 provided for each type of virus checker.

1 **[00091]** Weighted round-robin load balancing assigns virus scan requests to the
2 virus checkers so that the frequency at which the virus checker (i) receives virus checking
3 requests is proportional to the respective weight $W[i]$. Preferably the weighted round-
4 robin load balancing achieves a result substantially similar to conventional round-robin
5 load balancing for the case where the weights become substantially equal. This can be
6 done in a number of ways. One such way, described below with reference to FIGS. 16-
7 17, uses a randomization method. In the randomization method, a distribution template
8 having $W[i]$ entries for the virus checker (i) is created for each virus checker (i), the
9 distribution templates are concatenated to form a distribution list, the distribution list is
10 randomized, and then the distribution list is used to distribute the virus scan request to the
11 virus checkers. Another such way, described below with reference to FIG. 18, uses a
12 deterministic method in which each virus checker (i) has an accumulator incremented by
13 its respective weight $W[i]$ each time a request is assigned to a virus checker. The request
14 is assigned to the virus checker having a maximum accumulated value, and then the
15 accumulator for this virus checker is decremented by the sum of all the weights.

16 **[00092]** FIGS. 16-17 shows how a data mover may perform load balancing
17 based on statistics using the randomization method of round-robin load balancing. In a
18 first step 170, the data mover locates a pool of “N” virus checkers. Next, in step 171, the
19 data mover initializes a distribution template with “N” entries arranged in a round-robin
20 sequence; i.e., $[C_1, C_2, C_3, \dots, C_N]$, where C_i denotes an entry in the distribution template
21 indicating the i th virus checker. In step 172, the data mover uses the distribution
22 template as an initial distribution list for virus checking by issuing virus checking
23 requests sequentially to the virus checkers on the distribution list. This initial distribution

1 list is used until in step 173, the data mover receives heartbeat responses including a
2 respective utilization $\alpha[i]$ from each virus checker. In step 174, the data mover indexes
3 the mapping table with each utilization $\alpha[i]$ to obtain a corresponding weight $W[i]$ for
4 load balancing upon each virus checker (i). Execution continues from step 174 of FIG.
5 16 to step 175 of FIG. 17.

6 **[00093]** In step 175 of FIG. 17, if all of the weights are equal, then execution
7 branches to step 176. In step 176, the data mover initializes a distribution template with
8 “N” entries arranged in a round-robin sequence; i.e., $[C_1, C_2, C_3, \dots, C_N]$. Then in step
9 177, the data mover uses the distribution template as the distribution list for virus
10 checking by issuing virus checking requests sequentially to the virus checkers on the list.
11 This distribution list is used for issuing virus checking requests until the next heartbeat
12 interval. When the next heartbeat interval occurs, execution loops back from step 178 to
13 step 173 in FIG. 16.

14 **[00094]** In step 175 of FIG. 16, if all of the weights $W[i]$ are not equal, then
15 execution continues from step 175 to step 179. In step 179, the data mover creates a
16 distribution template $T[i]$ for each virus checker (i). The distribution template includes
17 $W[i]$ entries indicating the virus checker (i); i.e., $T[i] = [C_{i1}, C_{i2}, C_{i3}, \dots, C_{iW[i]}]$. Then in
18 step 180, the data mover creates a distribution list by concatenating the distribution
19 templates $T[i]$ and randomizing the positions of the entries in the distribution list.

20 **[00095]** For example, suppose that there are four virus checkers C_1, C_2, C_3, C_4 .
21 The first virus checker C_1 has a weight $W[1]=1$, the second virus checker C_2 has a weight
22 $W[2]=2$, the third virus checker C_3 has a weight $W[3]=3$, and the fourth virus checker C_4

1 has a weight $W[4]=4$. Then the distribution templates are $T[1]=[C_1]$, $T[2]=[C_2, C_2]$,
2 $T[3]=[C_3, C_3, C_3]$, and $T[4]=[C_4, C_4, C_4, C_4]$. The distribution templates are
3 concatenated to form the distribution list $[C_1, C_2, C_2, C_3, C_3, C_3, C_4, C_4, C_4, C_4]$. (The
4 distribution list can have at most $N * \text{MAXWEIGHT}$ entries, where N is the number of
5 virus checkers, and MAXWEIGHT is the maximum weight.) The distribution list is
6 randomized, for example, to produce $[C_4, C_3, C_4, C_2, C_3, C_1, C_4, C_2, C_4, C_3]$.
7 Randomization of the distribution list avoids pattern overloading that otherwise would be
8 caused by repetitive use of the distribution templates over the interval.

9 **[00096]** In order to randomize the distribution list, the number of entries in the
10 distribution list (N) is obtained. Then, for each entry (i) in the list, a number $\text{RND}(i)$
11 between 1 and N is selected by a pseudo-random number generator function, and the (i th)
12 entry is swapped with the $\text{RND}(i)$ th entry in the list.

13 **[00097]** From step 180, execution continues to step 181. In step 181, the data
14 mover issues virus checking requests sequentially to the virus checkers on the distribution
15 list. When the end of the distribution list is reached, the data mover re-randomizes the
16 distribution list, and then issue virus checking requests sequentially to the virus checkers
17 beginning at the start of the newly randomized distribution list. Randomization of the
18 distribution list for repetitive re-use during the heartbeat interval avoids pattern
19 overloading of the virus checkers. When the next heartbeat interval occurs, execution
20 loops back from step 182 to step 173 in FIG. 16.

21 **[00098]** FIG. 18 shows the alternative load balancing routine that uses an
22 accumulator $A[i]$ for each virus checker for performing the weighted round-robin load
23 balancing. In a first step 191, the data mover clears a respective accumulator $A[i]$ for

1 each of the N virus checkers ($i = 0$ to $N-1$). In step 192, the data mover sets an initial
2 weight $W[i]$ for each virus checker. For example, for the case where the weights in the
3 mapping table range from 1 to 4, the initial weight is a mean or median value such as 2.
4 In step 193, the data mover computes the sum of the weights $W[i]$ for all of the virus
5 checkers $i = 0$ to $N-1$. In step 194, if a virus scan is needed, then execution continues to
6 step 195. In step 195, the accumulator $A[i]$ for each virus checker is incremented by the
7 respective weight for the virus checker.

8 **[00099]** In step 196, the data mover finds the index (k) of an accumulator
9 having a maximum accumulated value. For example, the index (k) of the accumulator
10 having a maximum accumulated value is found by the following procedure:

11

```
12         k = 0;  
13         maxa = A[0];  
14         if (N = 1) then return;  
15         for (i = 1 to N-1) {  
16             if (A[i] > maxa) then {  
17                 k = i;  
18                 maxa = A[i];  
19             }  
20         }  
21
```

1 **[000100]** In step 197, the k th accumulator $A[k]$ is decremented by the sum of the
2 weights (from step 193). In step 198, the virus scanning task is assigned to the k th virus
3 checker, and then execution loops back to step 194.

4 **[000101]** In step 194, if a virus scan is not needed, then execution branches to
5 step 199. In step 199, unless a virus checker reports a new utilization, execution loops
6 back to step 194. Otherwise, when a virus checker reports a new utilization, execution
7 continues to step 200. In step 200, the data mover gets the new virus checker utilization
8 $\alpha[i]$. In step 201, the data mover indexes a mapping table with the new utilization $\alpha[i]$ to
9 obtain a corresponding weight $W[i]$. From step 201, execution loops back to step 193 to
10 re-compute the sum of the weights.

11 **[000102]** In the load balancing procedure of FIG. 18, when a virus scan is
12 needed, the accumulator for each virus checker is incremented by its respective weight in
13 step 195, the virus checker having the maximum accumulator value is selected in step
14 196, and the accumulator of the selected virus checker is decremented by the sum of the
15 weights in step 197. Therefore, the frequency of selecting each virus checker becomes
16 proportional to its respective weight.

17 **[000103]** In view of the above, there has been described a method of reducing
18 system overload conditions without the cost of additional processing units in a distributed
19 data processing system. Performance parameters are obtained for the distributed
20 processing units. The performance parameters include a utilization value of each
21 distributed processing unit. Respective weights are obtained for the distributed
22 processing units by applying a mapping function to the utilization values. The respective
23 weights are used for weighted round-robin load balancing of work requests upon the

1 distributed processing units. In one implementation, a set of utilization values of the
2 distributed processing units is collected in response to a periodic heartbeat signal, a set of
3 weights are produced from the set of utilization values, a distribution list is produced
4 from the set of weights, and the distribution list is randomized repetitively for re-use
5 during the heartbeat interval. In another implementation, an accumulator is maintained
6 for each distributed processing unit, and when a work request needs to be assigned, the
7 accumulator for each distributed processing unit is incremented by its respective weight,
8 a distributed processing unit having a maximum accumulator value is selected, and the
9 accumulator of the selected processing unit is decremented by the sum of the weights.

10